Implementación y Evaluación de un Agente de Ajedrez AlphaZero-like mediante MCTS y Redes Convolucionales en R

Anthonny Flores (y otros co-autores si aplica)
Escuela de Matemáticas
Universidad de Costa Rica
San José, Costa Rica
Email: tu.correo.electronico@ejemplo.com

Abstracto

El presente trabajo describe la implementación de un agente de inteligencia artificial para el juego de ajedrez, siguiendo la arquitectura seminal de **AlphaZero (Zero-like)**. La metodología se centra en el algoritmo de búsqueda **Monte Carlo Tree Search (MCTS)** guiado por una **Red Neuronal Profunda (NN)** que predice tanto la política (**P**) como el valor (**V**) de la posición. Se detalla la estructura del *tensor* de entrada ($8 \times 8 \times 18$) para la codificación FEN. El entrenamiento se realiza a través de un proceso de **Self-Play** (Auto-entrenamiento), donde el agente se enfrenta a sí mismo para generar datos de alta calidad. La evaluación se centra en la estabilidad de las simulaciones MCTS, la eficacia de las funciones de valor y política, y la capacidad de rendimiento del agente contra una IA externa (**Stockfish**). Finalmente, se reportan métricas clave de desempeño y tiempo de ejecución.

—

I. Motivación.

El ajedrez ha servido históricamente como un campo de pruebas crucial para la inteligencia artificial. Desde los sistemas basados en la búsqueda heurística de fuerza bruta, como **Deep Blue**, hasta el enfoque basado en el aprendizaje por refuerzo profundo de **AlphaZero**, la evolución en este dominio es notable.

La implementación de agentes de ajedrez Zero-like se motiva por tres puntos principales:

1. Exploración de MCTS Dirigido por NN: Superar las limitaciones de la búsqueda de fuerza bruta tradicional mediante un algoritmo de Monte Carlo Tree Search (MCTS). MCTS utiliza la exploración (U) y la explotación (Q) de la fórmula **UCT/PUCT** para centrar eficientemente la búsqueda, siendo dirigido por las predicciones de la Red Neuronal [1], [2]. 2. Aprendizaje por Refuerzo Profundo (DRL): El modelo aprende de la experiencia sin conocimiento humano previo (*tabula rasa*), utilizando partidas de **Auto-entrenamiento** para mejorar iterativamente la política y la función de valor. 3. Análisis de Rendimiento en R: Evaluar la viabilidad y el rendimiento de implementar algoritmos intensivos computacionalmente, como MCTS y la manipulación de tensores, utilizando el entorno **R**.

El proyecto busca implementar estos componentes clave para construir un agente capaz y analizar las métricas internas de su proceso de toma de decisiones.

—

II. Metodología.

A. Arquitectura del Agente

El agente implementado replica la estructura central de AlphaZero: un bucle iterativo de **MCTS** y una **Red Neuronal** (NN) que se entrenan mutuamente.

1. Red Neuronal (NN): Es el componente central entrenable. Toma la representación de la posición del tablero y produce dos salidas:
   - Política (**P**): Un vector de probabilidades sobre todos los movimientos legales posibles.
   - Valor (**V**): Un escalar que estima la probabilidad de que el jugador actual gane, variando entre $[-1, 1]$.

2. MCTS: Utiliza la Política (**P**) como *prior* para la exploración y el Valor (**V**) para respaldar las simulaciones, optimizando el coeficiente **C.PUCT** (Coeficiente de Exploración) en la fórmula de selección UCB.

B. Codificación del Estado (FEN a Tensor)

El estado del tablero se codifica a partir de la notación FEN en un *tensor* de entrada. La función fen.to.vector transforma el estado en una matriz de $8 \times 8 \times 18$.

- Planos 1-12: Representación One-Hot de las 6 piezas (P, N, B, R, Q, K) para ambos colores, en una cuadrícula 8×8.
- Planos 13-18: Información auxiliar: Turno del jugador (13), Opciones de enroque (14), Repeticiones (15), Capa constante (16), Reloj de medio movimiento (17), y Número de movimiento completo (18).

C. Entrenamiento por Auto-Juego (Self-Play)

El entrenamiento se realiza generando una gran cantidad de partidas donde el agente juega contra su propia versión más reciente (bot.vs.bot.game). Los hiperparámetros clave son:

- Simulaciones MCTS (NUM.SIMULATIONS): Determina la profundidad y calidad de la búsqueda en cada movimiento (3200 simulaciones).
- Ruido de Dirichlet: Se aplica a la política (**P**) en los primeros movimientos (num.moves < 30) para fomentar la exploración.
- Temperatura: Se utiliza una temperatura alta (temperature $= 1.0$) en las jugadas iniciales (num.moves < 15) para promover la diversidad, y una baja (0.01) en las finales para forzar la explotación del mejor movimiento.

Los datos generados incluyen el FEN, el vector $\mathbf{P}_{\text{MCTS}}$ (la política objetivo), y el valor final del juego (Z) como etiquetas de entrenamiento.

D. Métricas de Evaluación

La evaluación del agente se enfoca en dos aspectos: la calidad del MCTS y el rendimiento externo.

1. Métricas de MCTS Internas

- Valor Q(s,a) (Esperanza de Victoria): Se monitorea la diferencia entre el valor $\mathbf{V}_{\text{model}}$ (predicho por la NN) y el valor $Q(s, a)$ promedio (resultado de las simulaciones MCTS).
- Entropía de la Política ($\mathbf{P}_{\text{MCTS}}$): Mide la dispersión de probabilidades entre los movimientos legales.
- Tiempos de Ejecución: Se miden los tiempos de run.mcts para evaluar la eficiencia computacional de R en el bucle de simulación.

2. Evaluación Externa

- ELO: Se utiliza la función elo.function para calcular el ELO del agente.
- Partidas contra Stockfish: El agente se enfrenta a un motor de ajedrez de referencia (bot.vs.external.game) para obtener una evaluación objetiva de su fuerza de juego.

—

III. Resultados

A. Desempeño del Auto-Entrenamiento (Self-Play)

[Espacio para describir la evolución del ELO y la estabilidad del Valor (V) y la Política (P) a lo largo de las épocas de entrenamiento.]

B. Comparación MCTS vs. Stockfish

[Espacio para presentar los resultados de las partidas contra Stockfish, incluyendo el porcentaje de victorias, derrotas y empates, y el ELO final estimado.]

C. Análisis de Eficiencia (Tiempos de Ejecución)

[Espacio para reportar el tiempo promedio de run.mcts y la mediana del tiempo por jugada, comparando con el tiempo de respuesta de Stockfish (si aplica).]

—

IV. Discusión

A. Estabilidad y Convergencia

[Espacio para discutir si el modelo converge a una estrategia de juego estable, o si la política se mantiene demasiado exploratoria.]

B. Limitaciones del Entorno R

[Espacio para analizar el impacto de la naturaleza interpretada de R en los tiempos de MCTS y la manipulación de grandes tensores/datos.]

C. Estrategias Observadas

[Espacio para describir el estilo de juego del agente (agresivo, posicional) y las aperturas o finales que domina o ignora.]

—

V. Conclusiones

[Espacio para resumir los hallazgos más importantes sobre el éxito de la implementación AlphaZero en R y las principales fortalezas/debilidades del agente Hatchet1.]

—

Referencias

[1] Silver, D., Schrittwieser, J., Simonyan, K., et al. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354–359.

[2] Browne, C. B., Powley, E., Whitehouse, D., et al. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1–43.

[3] ... (Añadir referencias a la librería de ajedrez en R, TensorFlow/Keras si se usa, o literatura sobre codificación FEN.)